EN. 601.467/667 Introduction to Human Language Technology Deep Learning II

Shinji Watanabe

Today’s agenda

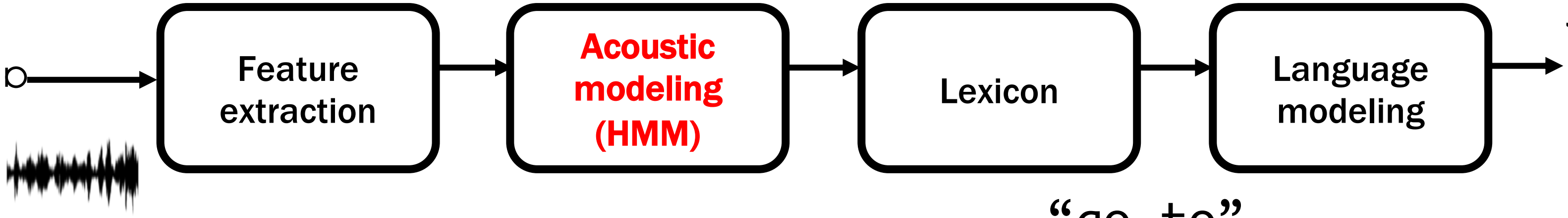
- Basics of (deep) neural network
- How to integrate DNN with HMM
- Recurrent neural network

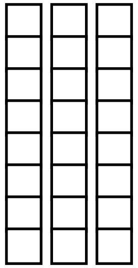
Speech recognition pipeline

G OW T UW

G OW Z T UW

"I want to **go to** Johns Hopkins campus"





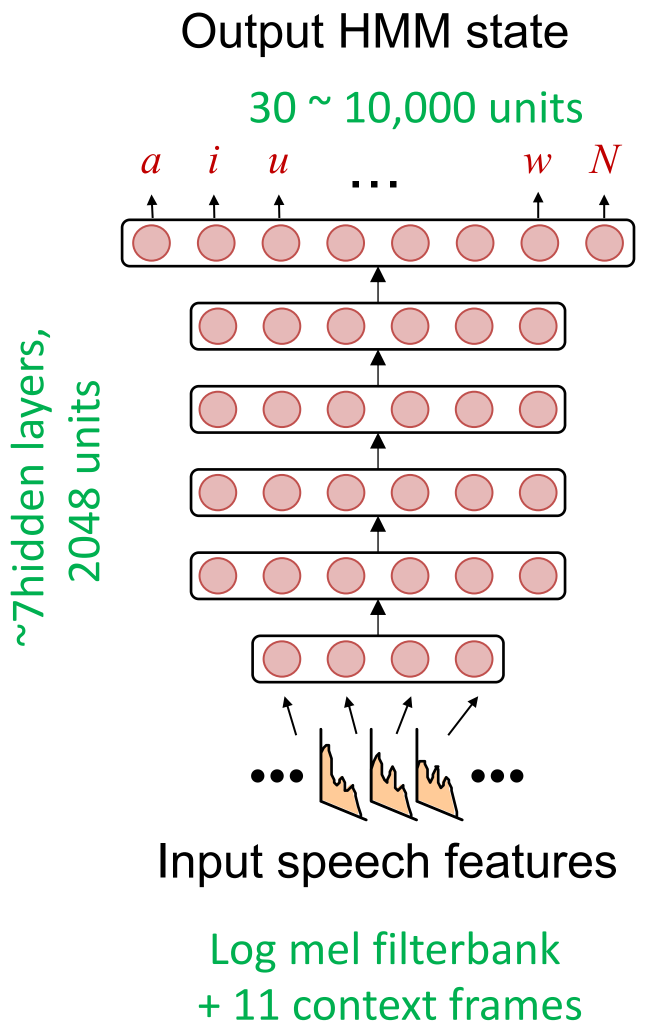
"go to"
"go two"
"go too"
"goes to"
"goes two"
"goes too"

$$\underset{W}{\operatorname{argmax}} \sum_{L} p(O|L)p(L|W)p(W)$$

$$p(O|L) \qquad p(L|W) \qquad p(W)$$

Feed-forward neural network for acoustic model

- Basic problem $p(s_t|\mathbf{o}_t)$
  - s_t: HMM state or phoneme
  - $\mathbf{o}_t$: speech feature vector
  - t: data sample
- Configurations
  - Input features
    - Context expansion
  - Output class
    - Softmax function
    - Training criterion
  - Number of layers
  - Number of hidden states
  - Type of non-linear activations



Input feature

- GMM/HMM formulation
  - Lot of conditional independence assumption and Markov assumption
  - Many of our trials are how to break these assumptions
- In GMM, we always have to care about the correlation
  - Delta, linear discriminant analysis, semi-tied covariance
- In DNN, we don't have to care ☺
  - We can simply concatenate
    the left and right contexts,
    and just throw it!

$$\mathbf{o}_t^{(2)} = \begin{bmatrix} \mathbf{o}_{t-r}^{(1)} \\ \vdots \\ \mathbf{o}_t^{(1)} \\ \vdots \\ \mathbf{o}_{t+r}^{(1)} \end{bmatrix}$$

Output

- Phoneme or HMM state ID is used
- We need to have a pair data of output and input data at frame t
  - First use the Viterbi alignment to obtain the state sequence

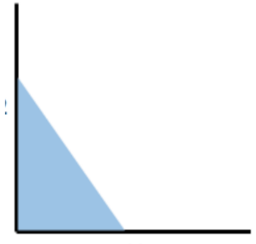
$$\hat{S} = \{\hat{s}_t | t = 1, \cdots, T\} = \underset{S}{\operatorname{argmax}}\, p(S, \mathbf{O} | \Theta_{\mathrm{gmm/hmm}})$$

  - Then, we get the input and output pair $\{\hat{s}_t, \mathbf{o}_t\}$ for all t
- Make acoustic model as a multiclass classification problem by predicting the all HMM state ID given the observation
  - Not consider any constraint in this stage (e.g., left to right, which is handled by an HMM during recognition)

Feed-forward neural networks

- Affine transformation and non-linear activation function (sigmoid function)

$$\mathbf{h}_t^{(1)} = \sigma(\mathbf{W}^{(1)}\mathbf{o}_t + \mathbf{b}^{(1)})$$



- Apply the above transformation L times

$$\mathbf{h}_t^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{h}_t^{(l-1)} + \mathbf{b}^{(l)})$$

- Softmax operation to get the probability distribution

$$\{p(s_t = j|\mathbf{o}_t)\}_{j=1}^{J} = \mathrm{softmax}(\mathbf{W}^{(L)}\mathbf{h}_t^{(L-1)} + \mathbf{b}^{(L)})$$

Linear operation

- Transforms $D^{(l-1)}$-dimensional input to $D^{(l)}$ output
$$f(\mathbf{h}^{(l-1)}) = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$$
    - $\mathbf{W}^{(l)} \in \mathbb{R}^{D^{(l)} \times D^{(l-1)}}$: Linear transformation matrix
    - $\mathbf{b}^{(l)} \in \mathbb{R}^{D^{(l)}}$: bias vector
- Derivatives
    - $\frac{\partial \sum_j w_{ij} h_j + b_i}{\partial b_{i'}} = \delta(i, i')$
    - $\frac{\partial (\sum_j w_{ij} h_j + b_i)}{\partial w_{i'j'}} = \delta(i, i') h_{j'}$

Sigmoid function

- Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

  - Convert the domain from $\mathbb{R}$ to $[0, 1]$
  - Elementwise sigmoid function:

$$\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}} = \left[\frac{1}{1 + e^{-x_d}}\right]_{d=1}^{D}$$

  - No trainable parameter in general
- Derivative
  - $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$

Softmax function

- Softmax function

$$p(j|\mathbf{h}) = [\mathrm{softmax}(\mathbf{h})]_j = \frac{e^{h_j}}{\sum_{i=1}^{J} e^{h_i}}$$

  - Convert the domain from $\mathbb{R}^J$ to $[0, 1]^J$ (make a multinomial dist. → classification)
  - Satisfy the sum to one condition, i.e., $\sum_{j=1} p(j|\mathbf{h}) = 1$
  - $J = 2$: sigmoid function
- Derivative
  - For $i = j$: $\frac{\partial p(j|\mathbf{h})}{\partial h_i} = p(j|\mathbf{h})(1 - p(j|\mathbf{h}))$
  - For $i \neq j$: $\frac{\partial p(j|\mathbf{h})}{\partial h_i} = -p(i|\mathbf{h})\, p(j|\mathbf{h})$
  - Or we can write as $\frac{\partial p(j|\mathbf{h})}{\partial h_i} = p(j|\mathbf{h})(\delta(i, j) - p(i|\mathbf{h}))$:$\delta(i, j)$: Kronecker's delta

What functions/operations we can use and cannot use?

- Most of elementary functions
  - $+, -, \times, \div, \log(\quad), \exp(\quad), \sin(\quad), \cos(\quad), \tan()$
- The function/operations that we cannot take a derivative, including some discrete operation
  - $\mathrm{argmax}_W p(W|O)$: Basic ASR operation, but we cannot take a derivative….
  - Discretization

Objective function design

- We usually use the cross entropy as an objective function

$$
\begin{aligned}
\mathcal{C}_{\mathrm{CE}}(\Theta_{\mathrm{dnn}}) &= \sum_t \mathrm{CE}[p^{\mathrm{ref}}(s_t) | p(s_t | \mathbf{o}_t, \Theta_{\mathrm{dnn}})] \\
&= -\sum_t \sum_{s_t} p^{\mathrm{ref}}(s_t) \log p(s_t | \mathbf{o}_t, \Theta_{\mathrm{dnn}})] \\
&= -\sum_t \sum_{s_t} \delta(s_t, \hat{s}_t) \log p(s_t | \mathbf{o}_t, \Theta_{\mathrm{dnn}})] \\
&= -\sum_t \log p(\hat{s}_t | \mathbf{o}_t, \Theta_{\mathrm{dnn}})]
\end{aligned}
$$

  - Since the Viterbi sequence is a hard assignment, the summation over states is simplified

Other objective functions

- Square error

$$\left|\mathbf{h}^{\mathrm{ref}} - \mathbf{h}\right|^2$$

  - We could also use p norm, e.g., L1 norm
- Binary cross entropy

$$\begin{aligned}\mathcal{C}_{\mathrm{Binary}}(\Theta_{\mathrm{dnn}}) &= -\sum_t \log p(\hat{s}_t|\mathbf{o}_t, \Theta_{\mathrm{dnn}})] \\ &= -\sum_t \log \sigma(\mathbf{h}_t)\end{aligned}$$

  - Again this is a special case of the cross entropy when the number of classes is two

Building blocks

Output: $s_t \in \{1, \ldots, J\}$

Softmax activation

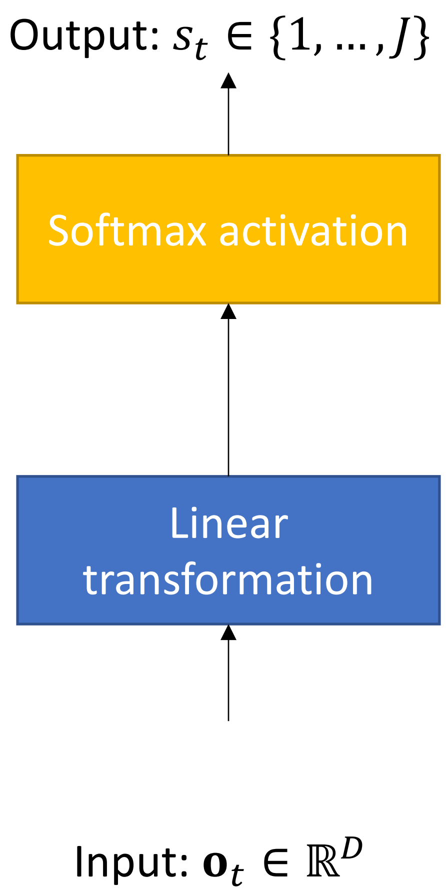
Sigmoid activation

Linear transformation

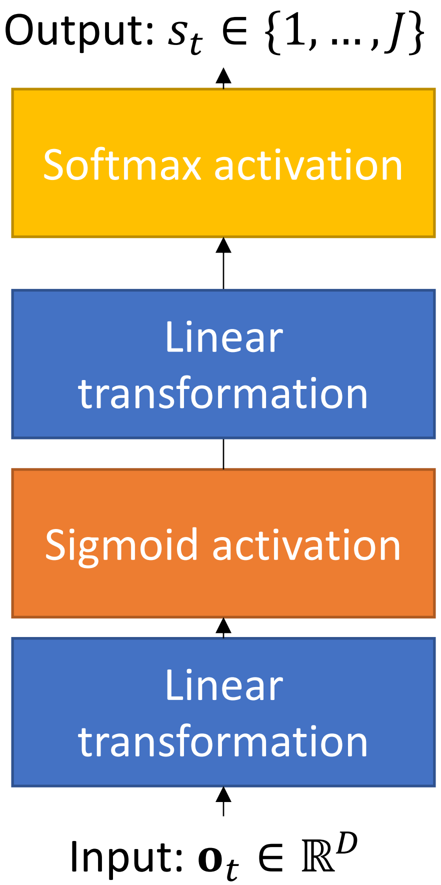
$+, -, \exp(\quad), \log(\quad), \text{etc.}$

Input: $\mathbf{o}_t \in \mathbb{R}^D$

Building blocks



Building blocks



Building blocks

Output: $s_t \in \{1, \ldots, J\}$

Softmax activation

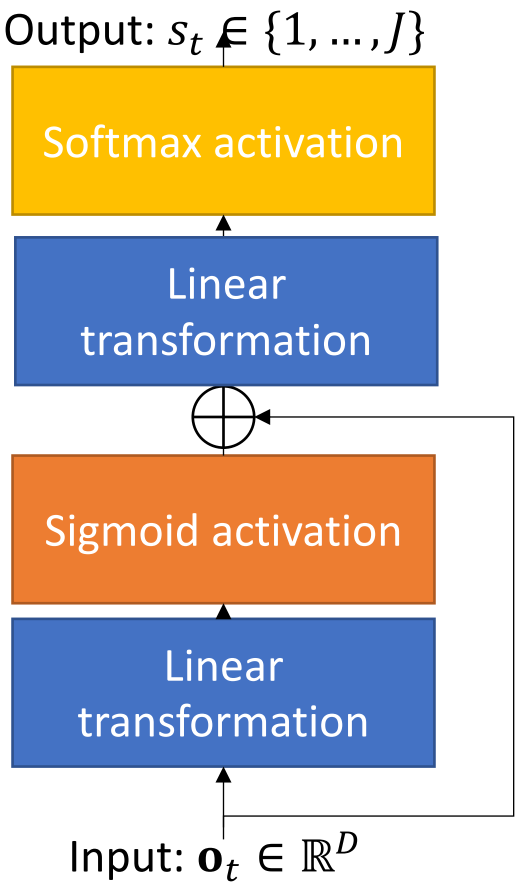
Sigmoid activation

Linear transformation

$+, -, \exp(\quad), \log(\quad)$, etc.

Input: $\mathbf{o}_t \in \mathbb{R}^D$

Building blocks



How to optimize? Gradient decent and their variants

- Take a derivative and update parameters with this derivative

$$\Theta_{\mathrm{dnn}}^{(\mathrm{new})} = \Theta_{\mathrm{dnn}}^{(\mathrm{old})} - \rho \frac{\partial}{\partial \Theta_{\mathrm{dnn}}} \mathcal{C}_{\mathrm{CE}}(\Theta_{\mathrm{dnn}}) \Big|_{\Theta_{\mathrm{dnn}} = \Theta_{\mathrm{dnn}}^{(\mathrm{old})}}$$

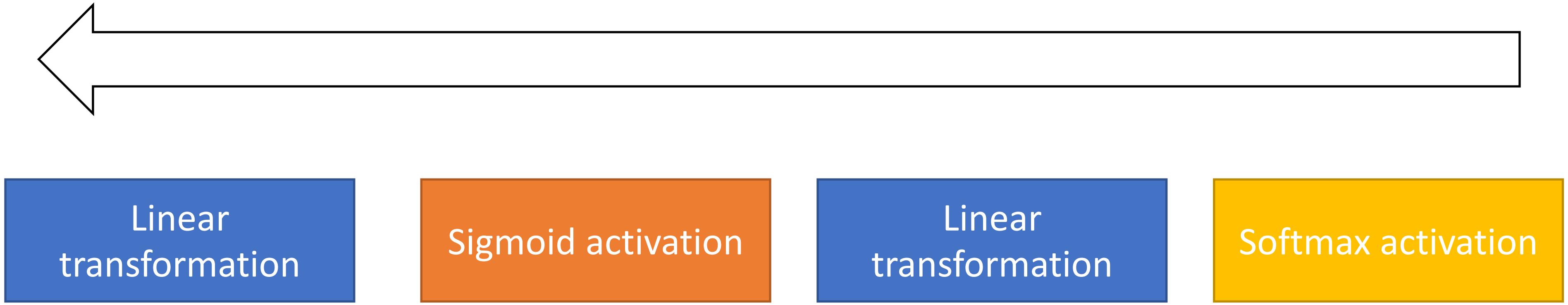
- Chain rule

$$\frac{\partial}{\partial \theta} f(g(\theta)) = \frac{\partial}{\partial g} \frac{\partial g}{\partial \theta} f(g(\theta)) = f'(g(\theta)) g'(\theta)$$

- Learning rate ρ

Deep neural network: nested function

- Chain rule to get a derivative recursively
  - Each transformation (Affine, sigmoid, and softmax) has analytical derivatives and we just combine these derivatives
  - We can obtain the derivative from the back propagation algorithm



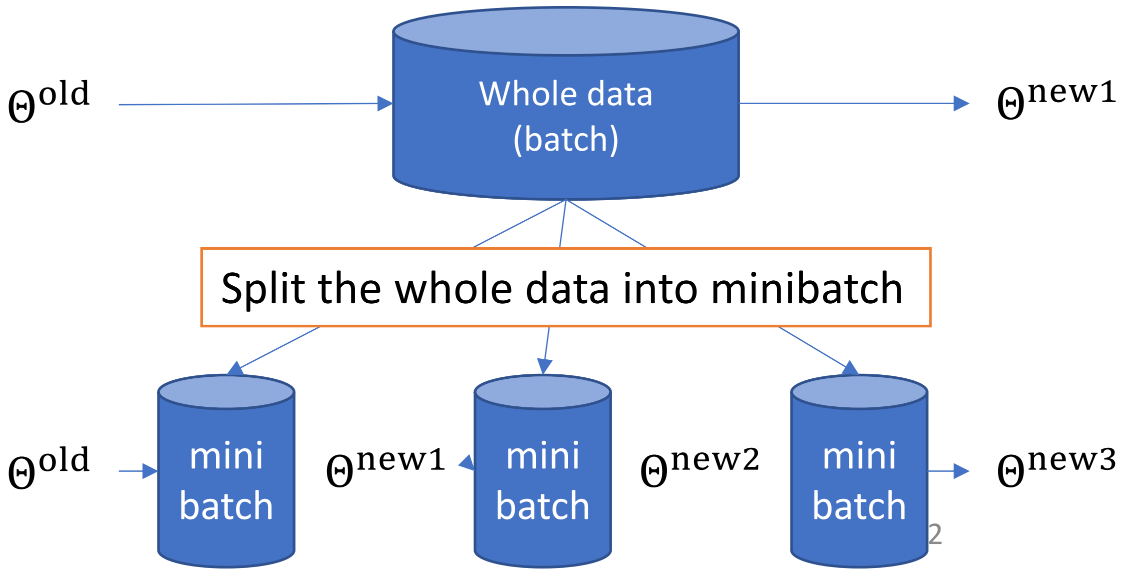
Minibatch processing

- Batch processing
  - Slow convergence
  - Effective computation
- Online processing
  - Fast convergence
  - Very inefficient computation
- Minibatch processing
  - Something between batch and online processing

$$\Theta_{\text{dnn}}^{(\text{new})} = \Theta_{\text{dnn}}^{(\text{old})} - \rho \frac{\partial}{\partial \Theta_{\text{dnn}}} \mathcal{C}_{\text{CE}}(\Theta_{\text{dnn}}) \Big|_{\Theta_{\text{dnn}} = \Theta_{\text{dnn}}^{(\text{old})}}$$

where

$$\mathcal{C}_{\text{CE}}(\Theta_{\text{dnn}}) = -\boxed{\sum_t} \log p(\hat{s}_t | \mathbf{o}_t, \Theta_{\text{dnn}})$$



How to set ρ?

$$\Theta^{(\tau+1)} = \Theta^{(\tau)} - \rho \cdot \Delta_{\text{grad}}^{(\tau)}$$

- Stochastic Gradient Decent (SGD)
  - Use a constant value (hyper-parameter)
  - Can have some heuristic tuning (e.g., $\rho \leftarrow 0.5 \times \rho$ when the validation loss started to be degraded. Then the decay factor becomes another a hyperparameter)
- Adam, AdaDelta, RMSProp, etc.
  - Use current or previous gradient information adaptively update $\rho(\Delta_{\text{grad}}^{(\tau)}, \Delta_{\text{grad}}^{(\tau-1)}, \dots)$
  - Still has hyperparameters to make a balance between current and previous gradient information
- Choice of an appropriate optimizer and its hyperparameters is critical

Today’s agenda

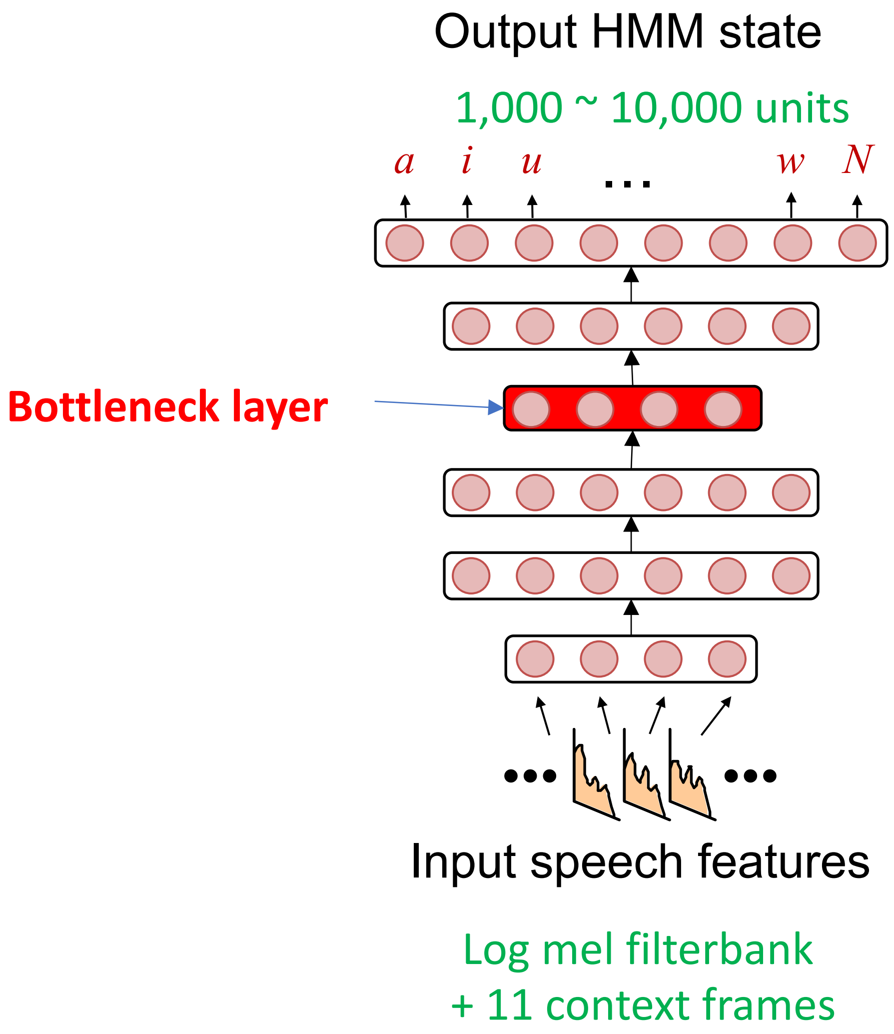
- Basics of (deep) neural network
- How to integrate DNN with HMM
- Recurrent neural network

How to integrate DNN with HMM

- Bottleneck feature
- DNN/HMM hybrid

Bottleneck feature

- Train DNN, but one layer having a narrow layer
- Use a hidden state vector for GMM/HMM
- Nonlinear feature extraction with discriminative abilities
- Can combine with existing GMM/HMM



DNN/HMM hybrid

- How to make it fit to the HMM framework?
  - Use the Bayes rule to convert the posterior to the likelihood

$$p(\mathbf{o}_t|s_t) \to p(s_t|\mathbf{o}_t, \Theta_{\text{dnn}})/p(s_t)$$

  - $p(s_t)$ is obtained by the maximum likelihood (unigram count)
- Need a modification in the Viterbi algorithm during recognition

Today’s agenda

- Basics of (deep) neural network
- How to integrate DNN with HMM
- Recurrent neural network

Recurrent neural network

- Basic problem
  - HMM state (or phoneme) or speech feature is a sequence

$$s_1, s_2, \dots, s_\mathrm{t}$$
$$\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_t$$

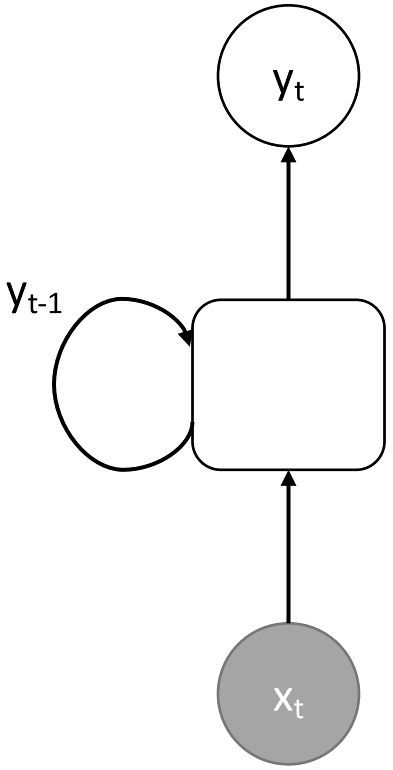
  - It's better to consider context (e.g., previous input) to predict the probability of s_t

$$p(s_t|\mathbf{o}_t) \rightarrow p(s_t|\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_t)$$

- Recurrent neural network (RNN) can handle such problems

Recurrent neural network (Elman type)

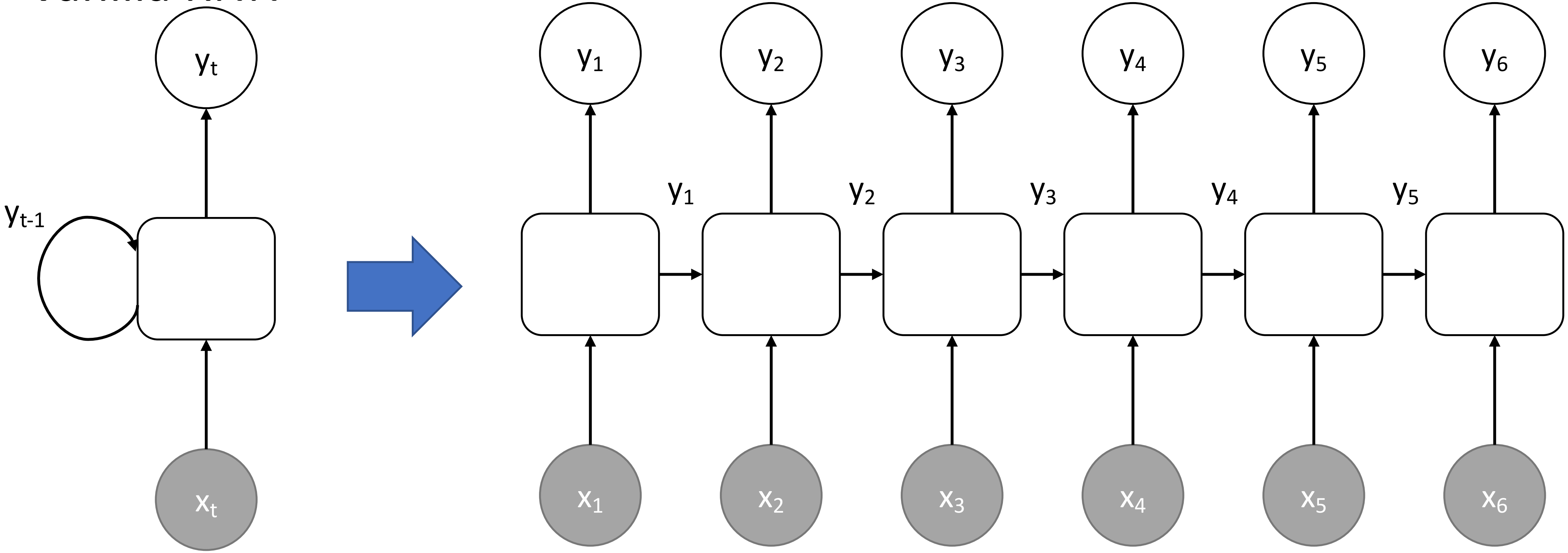
- Vanilla RNN: We ignore the bias term for simplicity



$$\mathbf{y}_t = \sigma\left(\mathbf{W}\begin{bmatrix}\mathbf{y}_{t-1}\\ \mathbf{x}_\mathrm{t}\end{bmatrix}\right)$$

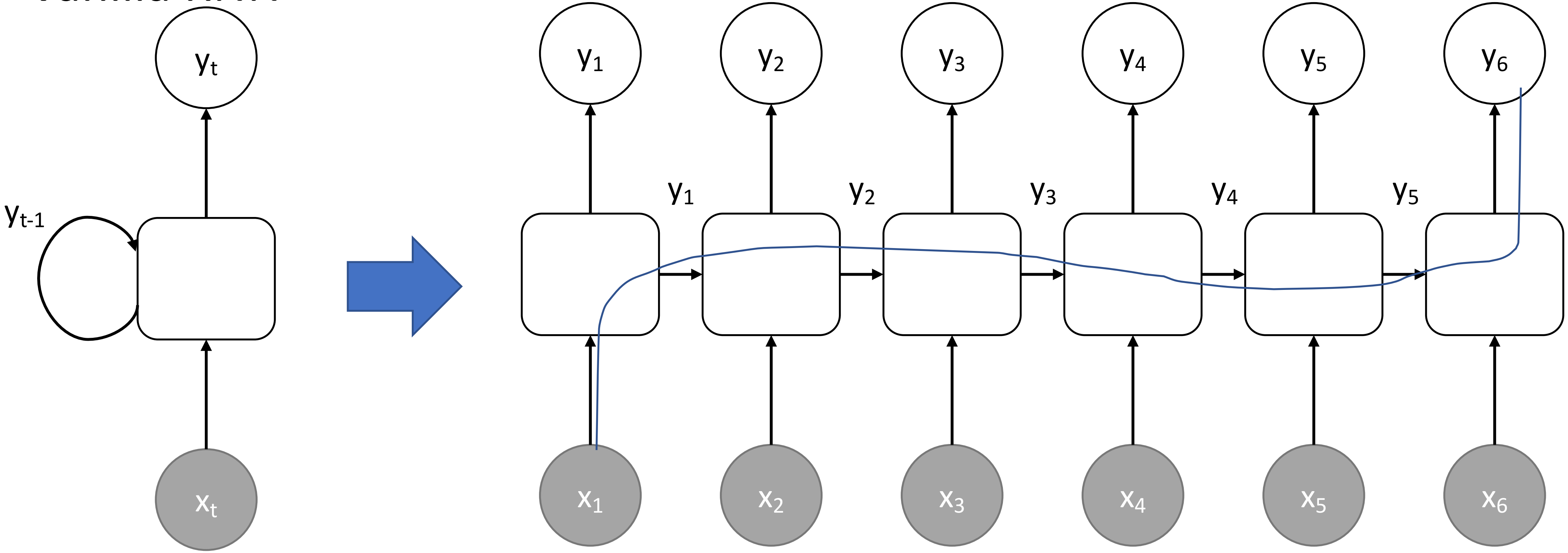
Recurrent neural network (Elman type)

- Vanilla RNN



Recurrent neural network (Elman type)

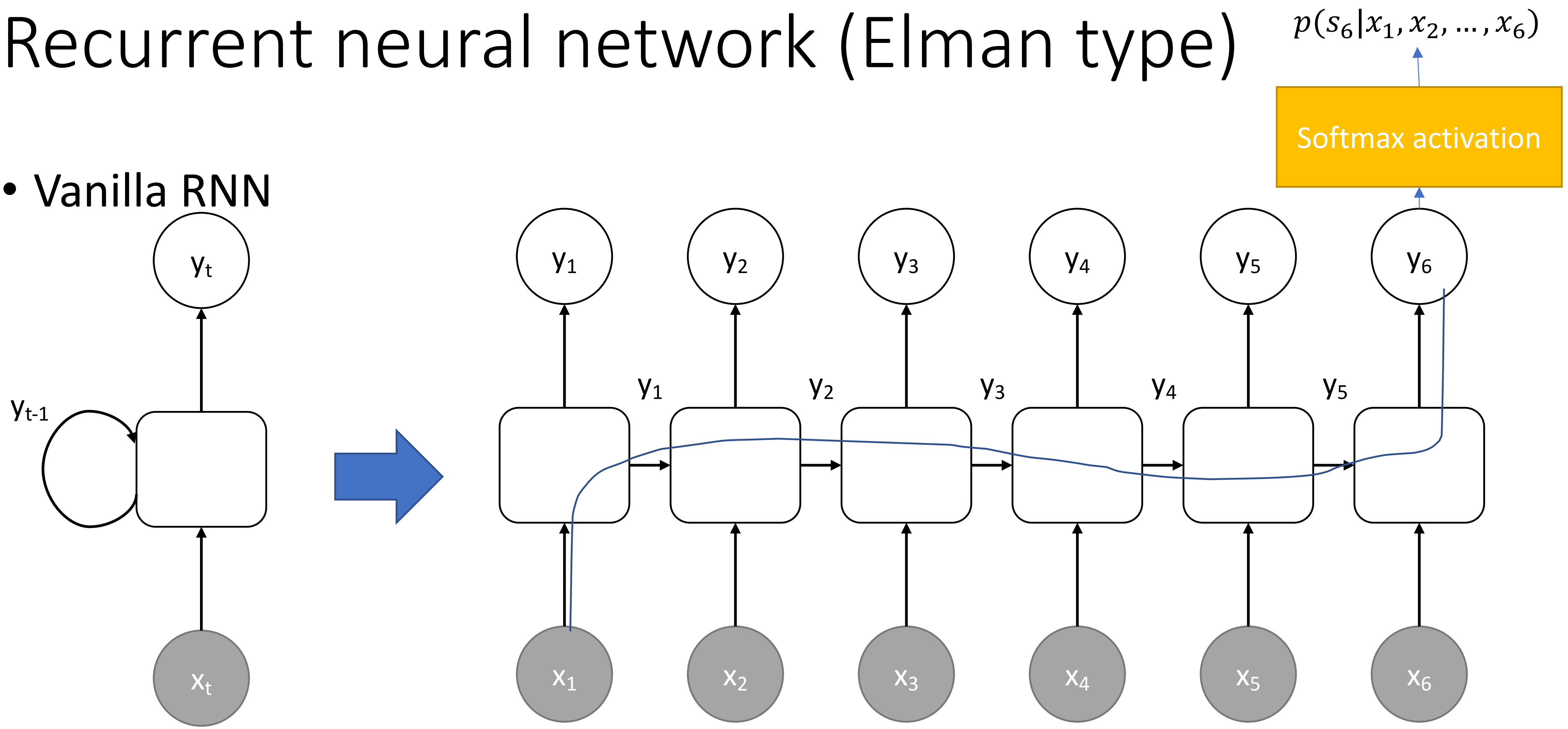
- Vanilla RNN



Possibly consider long-range effect (but longer weaker)
no future context

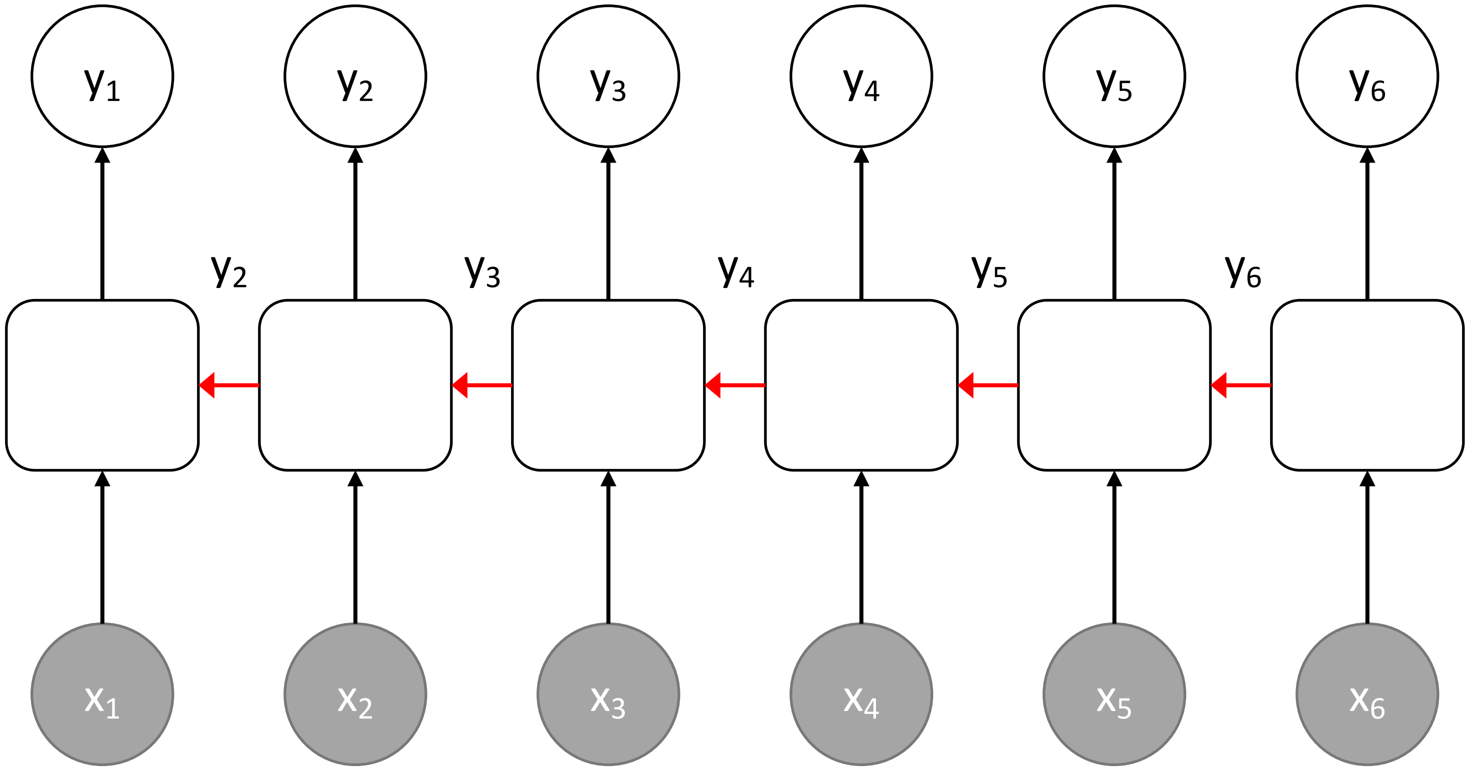
Recurrent neural network (Elman type)

- Vanilla RNN



We can compute the posterior distribution $p(s_t|\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t)$

Bidirectional RNN



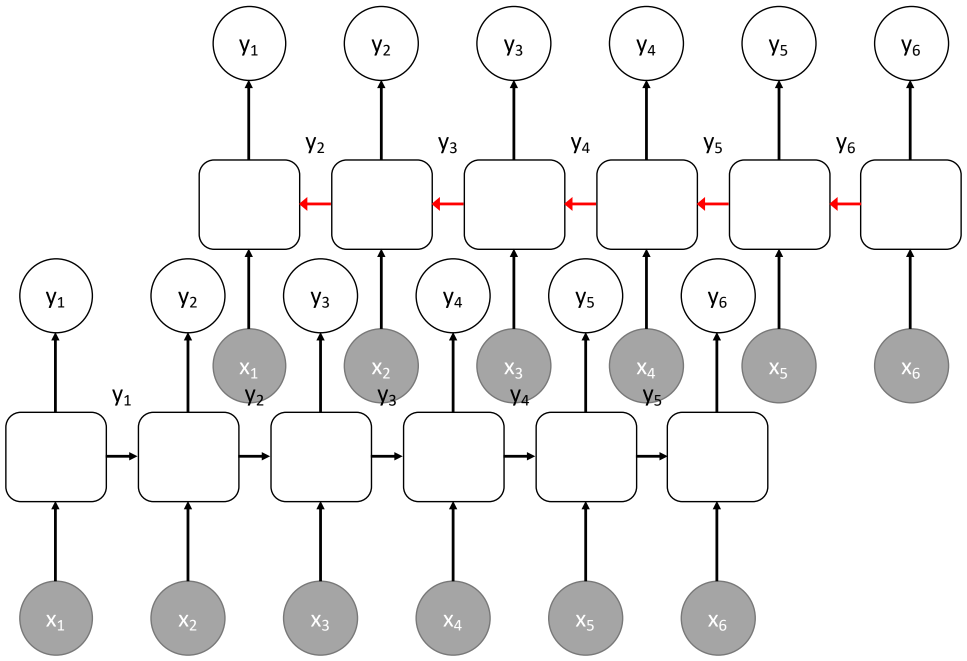
We can compute the posterior distribution $p(s_t | \mathbf{o}_t, \mathbf{o}_{t+1}, \ldots,)$

Bidirectional RNN

$$\mathbf{y}_t = \begin{bmatrix} \overrightarrow{\mathbf{y}}_{t+1} \\ \overleftarrow{\mathbf{y}}_{t+1} \end{bmatrix}$$

$$\overleftarrow{\mathbf{y}_t} = \sigma\left(\mathbf{W}\begin{bmatrix} \overleftarrow{\mathbf{y}_{t-1}} \\ \mathbf{x}_\mathrm{t} \end{bmatrix}\right)$$

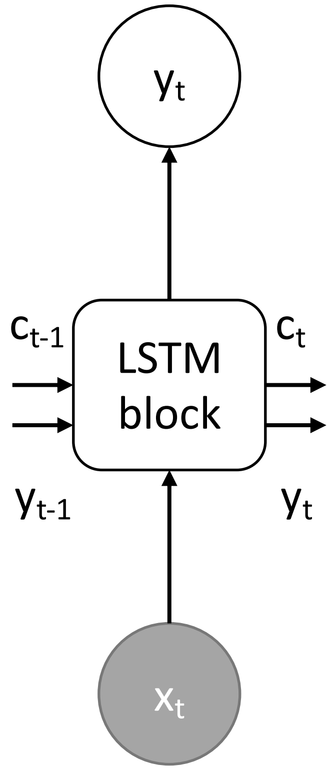
$$\overrightarrow{\mathbf{y}_t} = \sigma\left(\mathbf{W}\begin{bmatrix} \overrightarrow{\mathbf{y}_{t-1}} \\ \mathbf{x}_\mathrm{t} \end{bmatrix}\right)$$



We can compute the posterior distribution $p(s_t|\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_t, \mathbf{o}_{t+1}, \dots,)$

Long short-term memory RNN

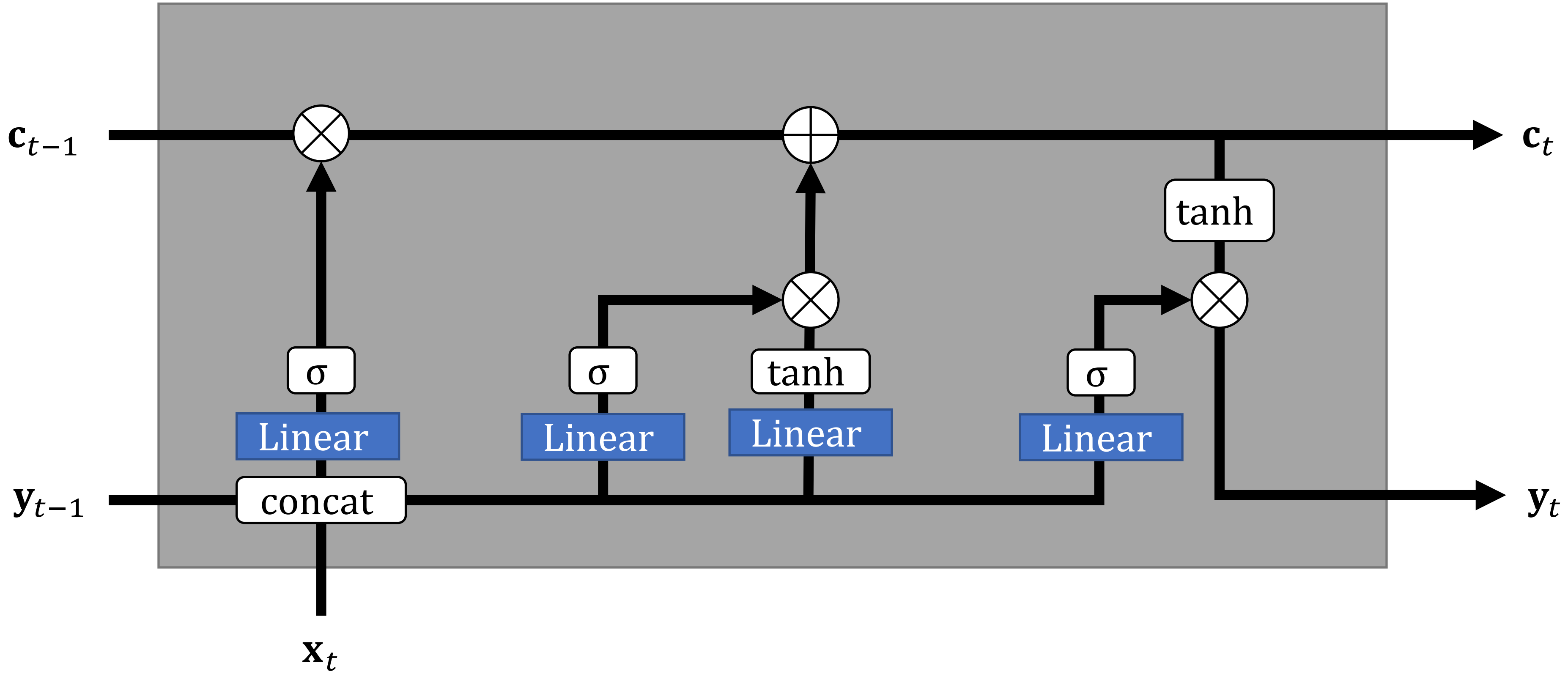
- Keep two states
  - Normal recurrent state: y_t
  - Memory cell: c_t



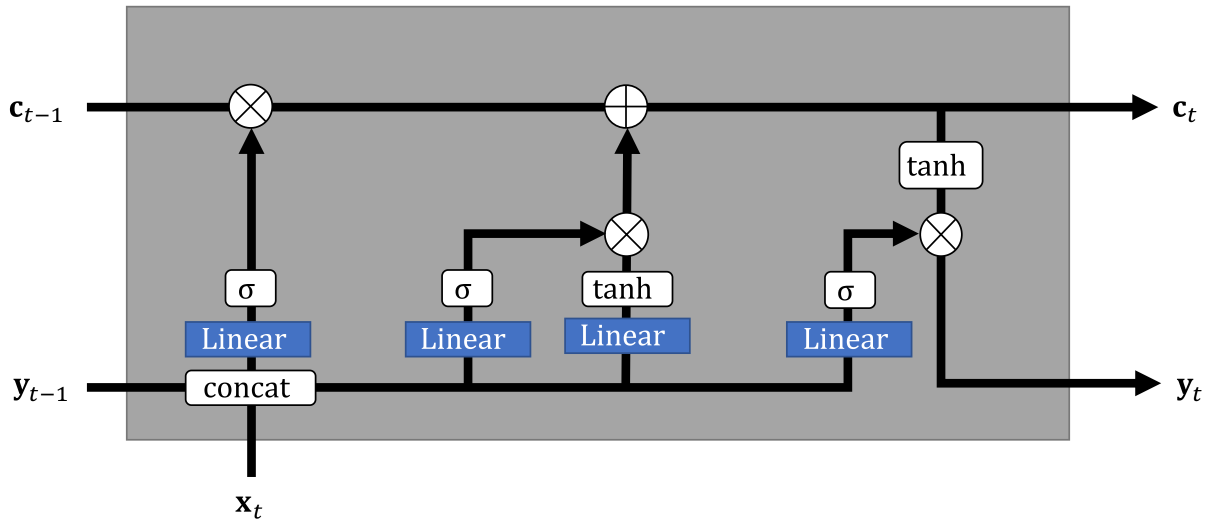
LSTM block

Operations **w/o** trainable parameters

Operations **w/** trainable parameters



LSTM block



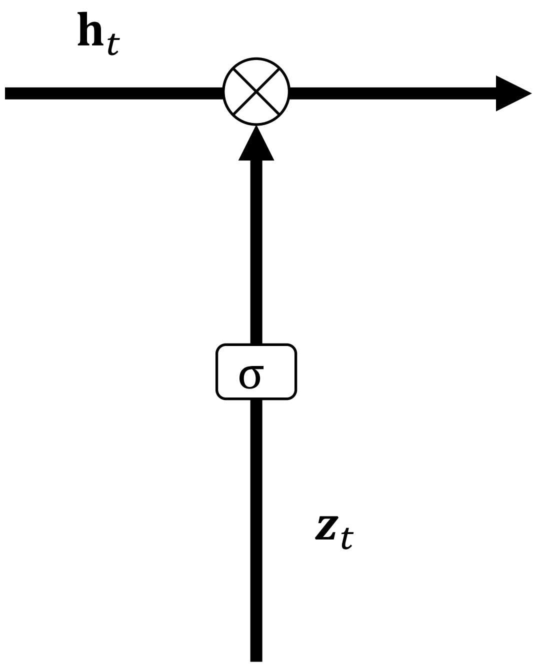
- Cell state keeps the history information
  1. It **will be** forgotten
  2. New information from x_t **will be** added
  3. The cell information **will be** outputted as y_t

- ”will be” function is implemented by a gate function [0, 1] through the sigmoid activation

tanh and sigmoid activations

- sigmoid
  - Convert the domain from $\mathbb{R}$ to $[0, 1]$
  - Used as a **gating** (weight the state vector (information))

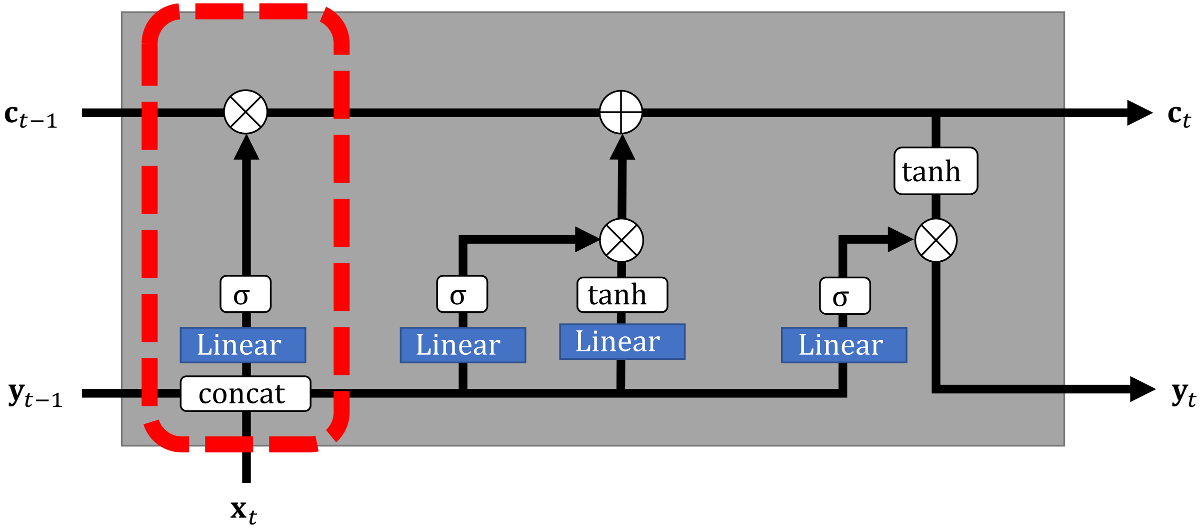
$$\mathbf{h}_t \otimes \sigma(\mathbf{z}_t)$$



- tanh
  - Convert the domain from $\mathbb{R}$ to $[-1, 1]$
  - Allow negative and positive values

$$\tanh(\mathbf{x}) = \frac{e^{\mathbf{x}} - e^{-\mathbf{x}}}{e^{\mathbf{x}} + e^{-\mathbf{x}}}$$

LSTM block

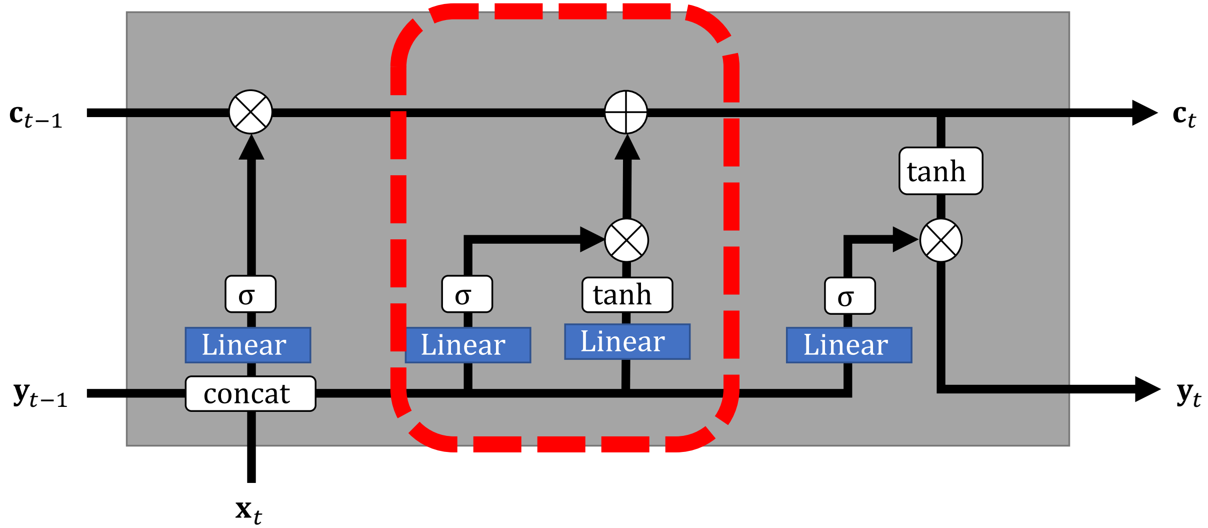


- Cell state keeps the history information
    1. It **will be** forgotten
    2. New information from x_t **will be** added
    3. The cell information **will be** outputted as y_t

$$\mathbf{g}_{\text{forget}} = \sigma\left(\mathbf{W}_{\text{forget}}\begin{bmatrix}\mathbf{y}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b}_{\text{forget}}\right)$$

$$\mathbf{c}_{t-1} \otimes \mathbf{g}_{\text{forget}}$$

LSTM block

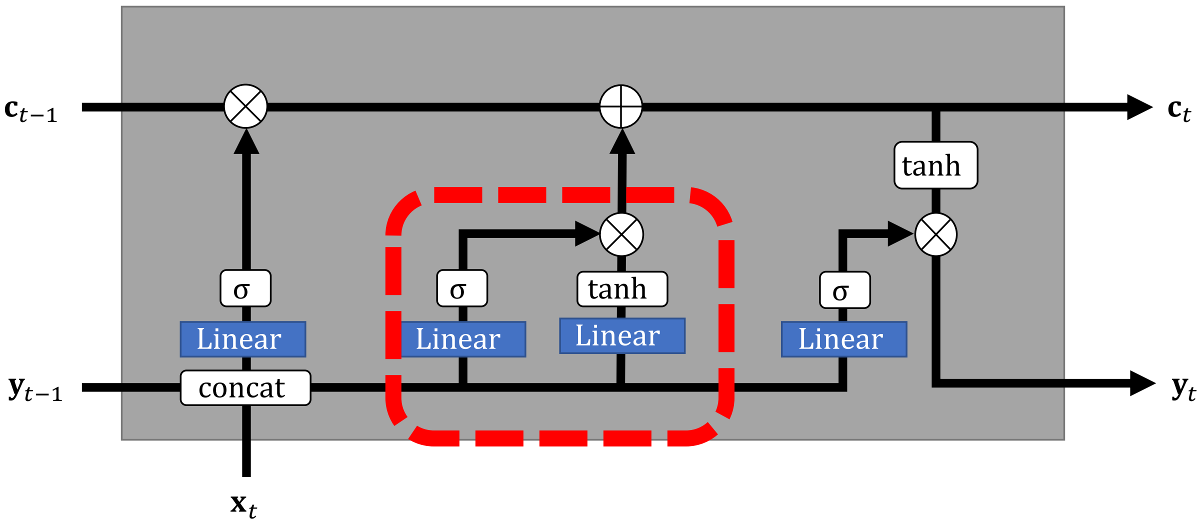


- Cell state keeps the history information
  1. It **will be** forgotten
  2. New information from x_t **will be** added
  3. The cell information **will be** outputted as y_t

$$\mathbf{g}_{\text{input}} = \sigma\left(\mathbf{W}_{\text{input}} \begin{bmatrix} \mathbf{y}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{\text{input}}\right)$$

$$\tanh\left(\mathbf{W}_{\text{cell}} \begin{bmatrix} \mathbf{y}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{\text{cell}}\right) \otimes \mathbf{g}_{\text{input}}$$

LSTM block

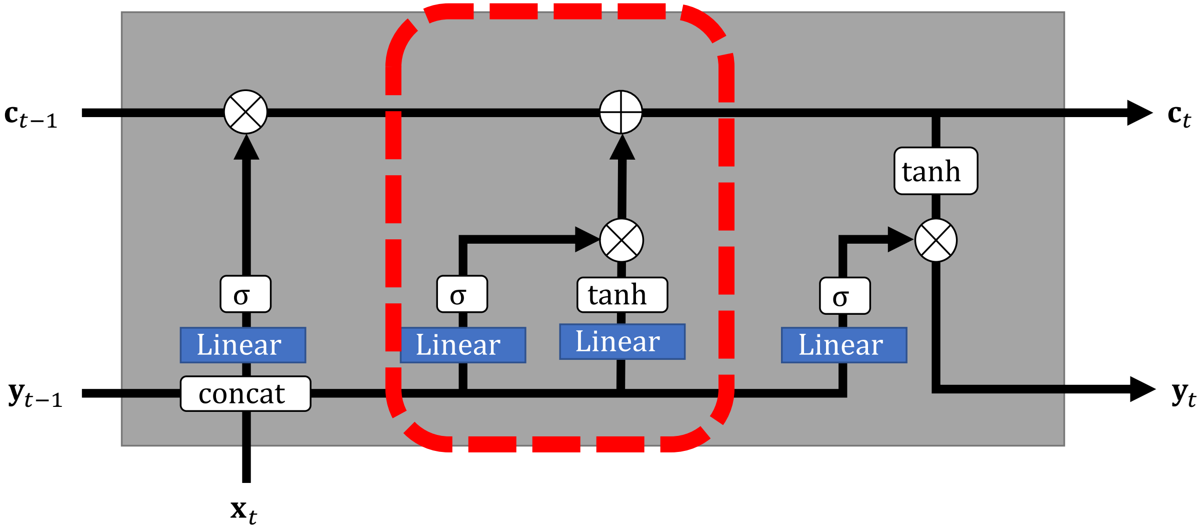


- Cell state keeps the history information
  1. It **will be** forgotten
  2. New information from x_t **will be** added
  3. The cell information **will be** outputted as y_t

$$\mathbf{g}_{\text{input}} = \sigma\left(\mathbf{W}_{\text{input}}\begin{bmatrix}\mathbf{y}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b}_{\text{input}}\right)$$

$$\tanh\left(\mathbf{W}_{\text{cell}}\begin{bmatrix}\mathbf{y}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b}_{\text{cell}}\right) \otimes \mathbf{g}_{\text{input}}$$

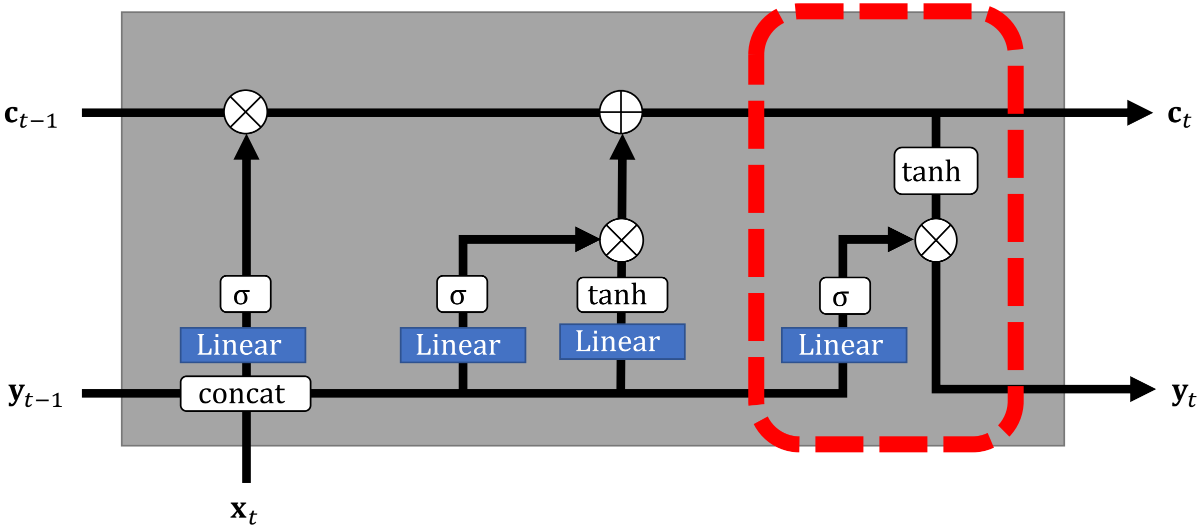
LSTM block



- Cell state keeps the history information
  1. It **will be** forgotten
  2. New information from x_t **will be** added
  3. The cell information **will be** outputted as y_t

$$\mathbf{c}_t = \mathbf{c}_{t-1} \otimes \mathbf{g}_{\text{forget}} + \tanh\left(\mathbf{W}_{\text{cell}} \begin{bmatrix} \mathbf{y}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{\text{cell}}\right) \otimes \mathbf{g}_{\text{input}}$$

LSTM block

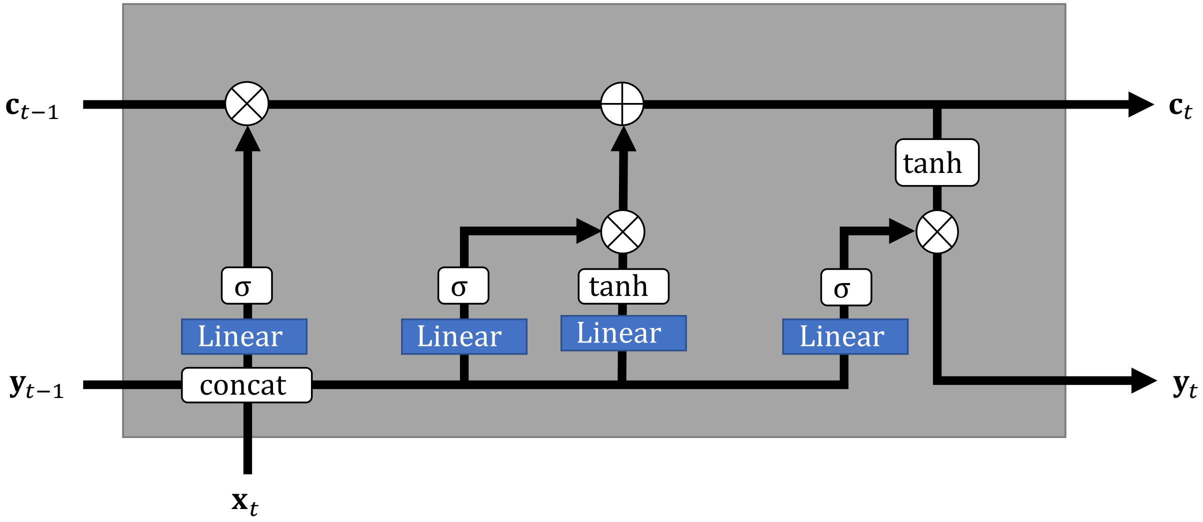


- Cell state keeps the history information
  1. It **will be** forgotten
  2. New information from x_t **will be** added
  3. The cell information **will be** outputted as y_t

$$\mathbf{g}_{\text{output}} = \sigma\left(\mathbf{W}_{\text{output}}\begin{bmatrix}\mathbf{y}_{t-1}\\ \mathbf{x}_t\end{bmatrix} + \mathbf{b}_{\text{output}}\right)$$

$$\mathbf{y}_t = \tanh(\mathbf{c}_t) \otimes \mathbf{g}_{\text{output}}$$

LSTM block summary



- 3 gating functions

$$\mathbf{g}_{\text{forget}} = \sigma\left(\mathbf{W}_{\text{forget}} \begin{bmatrix} \mathbf{y}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{\text{forget}}\right)$$

$$\mathbf{g}_{\text{input}} = \sigma\left(\mathbf{W}_{\text{input}} \begin{bmatrix} \mathbf{y}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{\text{input}}\right)$$

$$\mathbf{g}_{\text{output}} = \sigma\left(\mathbf{W}_{\text{output}} \begin{bmatrix} \mathbf{y}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{\text{output}}\right)$$

- Cell update

$$\mathbf{c}_t = \mathbf{c}_{t-1} \otimes \mathbf{g}_{\text{forget}} + \tanh\left(\mathbf{W}_{\text{cell}} \begin{bmatrix} \mathbf{y}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{\text{cell}}\right) \otimes \mathbf{g}_{\text{input}}$$

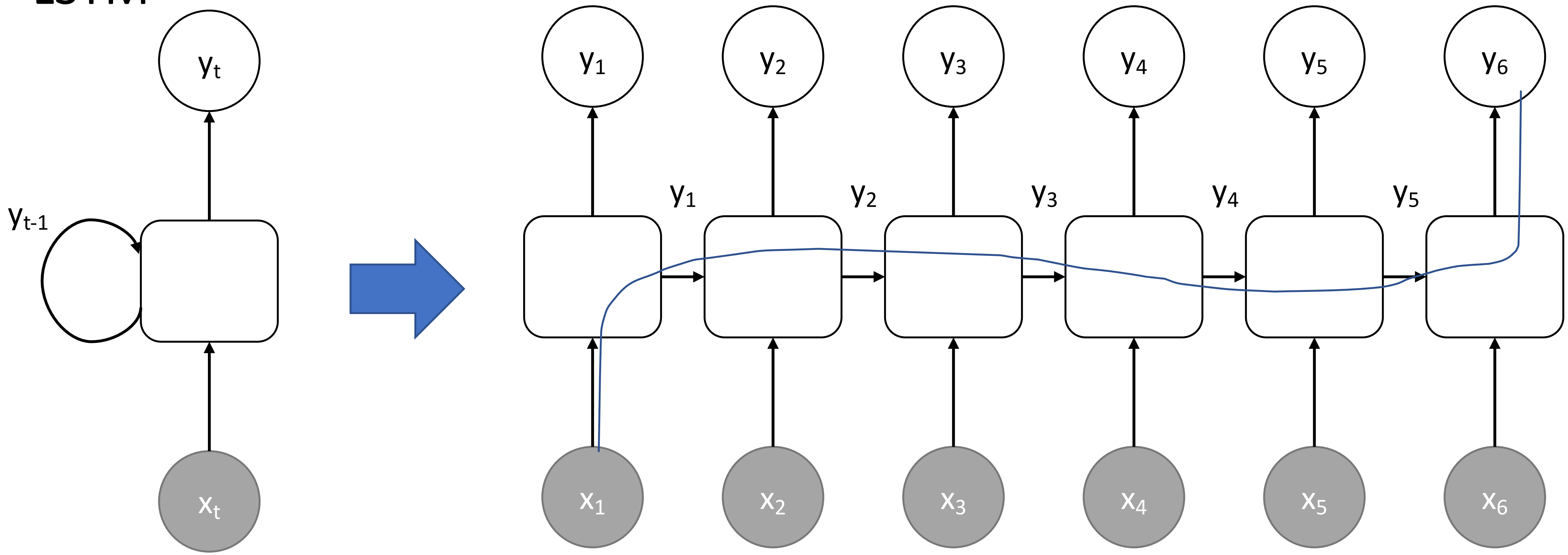
- Hidden state update

$$\mathbf{y}_t = \tanh(\mathbf{c}_t) \otimes \mathbf{g}_{\text{output}}$$

LSTM RNN

$$\mathbf{c}_t = \mathbf{c}_{t-1} \otimes \mathbf{g}_{\text{forget}} + \tanh\left(\mathbf{W}_{\text{cell}} \begin{bmatrix} \mathbf{y}_{t-1} \\ \mathbf{x}_t \end{bmatrix} + \mathbf{b}_{\text{cell}}\right) \otimes \mathbf{g}_{\text{input}}$$

- LSTM

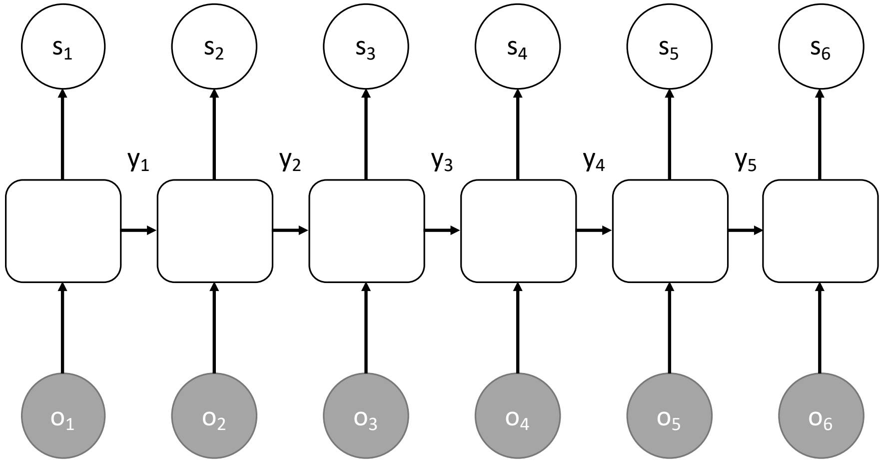


Possibly consider to keep the above initial information dependency by $\mathbf{g}_{\text{forget}} = \mathbf{1}$ and $\mathbf{g}_{\text{input}} = \mathbf{0}$ at $t > 1$

RNN can be used for both acoustic and language models

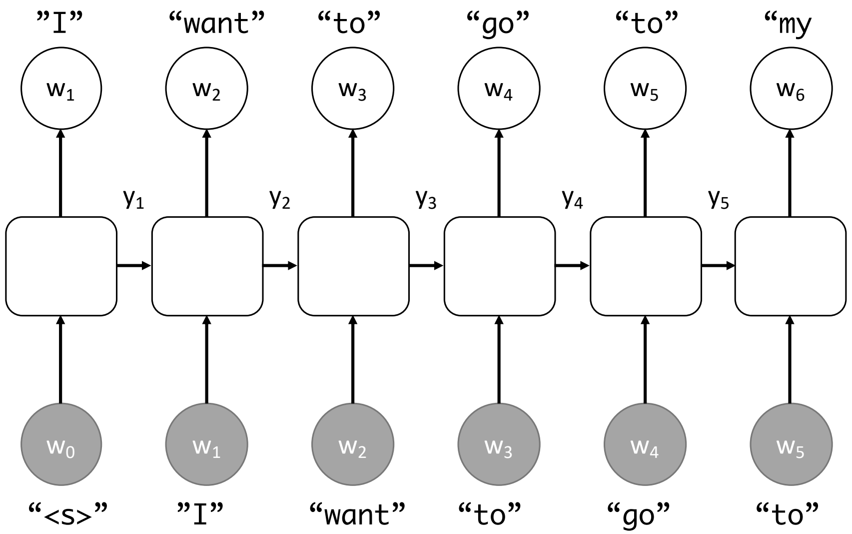
- HMM/DNN Acoustic model

$$p(s_t|\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_t)$$



- Language model

$$p(w_i|w_1, w_2, \ldots, w_{i-1})$$



Summary of today's talk

- Basics of deep neural network
  - Input, output, function block, back propagation, optimization
- Recurrent neural network
  - Now we can handle a sequence $(s_1, s_2, \ldots)$, $(w_1, w_2, \ldots)$, $(\mathbf{o}_1, \mathbf{o}_2, \ldots)$
- Integrate deep neural network for speech recognition
  - GMM/HMM → DNN/HMM or RNN/HMM
  - RNN language model

- Next lecture will extend this sequence handling function to directly model whole speech recognition in an ***end-to-end manner***